Free-ordered CUG on Chemical Abstract Machine

Satoshi Tojo
Mitsubishi Research Institute, Inc.
(e-mail) tojo@mri.co.jp

Abstract

We propose a paradigm for concurrent natural language generation. In order to represent grammar rules distributively, we adopt categorial unification grammar (CUG) where each category owns its functional type. We augment typed lambda calculus with several new combinators, to make the order of λ-conversions free for partial / local processing. The concurrent calculus is modeled with *Chemical Abstract Machine*. We show an example of a Japanese *causative* auxiliary verb that requires a drastic rearrangement of case domination.

1 Introduction

Parallel and distributed computation is expected to be the main stream of information processing. In the conventional generation, the rules for composition are given from the outside and those rules control all the behavior of the symbols or the objects, for assembling a hierarchical tree structure. For example, all the linguistic objects, such as words and phrases must be applied to so-called grammar rules to form grammatical structures or rational semantic representations, under a strict controller process. However, this kind of formalization obviously contradicts the partial / distributed processing that would be required in parallel architecture in future.

In order to represent grammar rules distributively, we adopt categorial grammar, where we can an attach local grammar rule to each word and phrase. What we aim in this paper is to propose a paradigm that enables partial / local generation through decompositions and reorganizations of tentative local structures.

In the following section, we introduce the extended λ-calculus. Thereafter we introduce the ChAM model and we reinterpret the model in terms of natural language processings. Then we show the model of *membrane* interaction model with the example of Japanese *causative* sentence that requires drastic change of domination of cases. Finally we will discuss the future of the model.

2 Extended typed λ-calculus

CUG (Categorial Unification Grammar) [8] is advantageous, compared to other phrase structure grammars, for parallel architecture, because we can regard categories as functional types and we can represent grammar rules locally. This means that we do not need externally-given grammar rules but those rules reside within each word or each phrase. In this section, we regard categories as polymorphic types and consider the type calculus. In later sections we denote categories by DAG (directed acyclic graph) of PATR grammar [5].

2.1 λ-calculus of polymorphic type

We use greek letters, for type schemas. For type constants we use $\sigma, \tau, \cdots$ while for type variables we use $\alpha, \beta, \cdots$. $a : \alpha$ represents that the object a is of type α. If α and β are types, then $\alpha \rightarrow \beta$ is a type.

The purpose of type inference is to infer the type of an object from a set of objects whose types are known. We presuppose that two type variables α and β are unified with a unifier θ. We use Γ for this set of type-known objects. The most important two rules are as follows: [1]

$$\frac{\Gamma\theta_1 \cup \{x : \alpha\theta_1\} \vdash t : \beta}{\Gamma\theta_1 \vdash \lambda x^{\alpha}.t : \alpha\theta_1 \rightarrow \beta} \tag{1}$$

$$\frac{\Gamma\theta_2\theta_3\theta_4 \vdash t : \alpha\theta_4 \rightarrow \beta\theta_4 \qquad \Gamma\theta_2\theta_3\theta_4 \vdash s : \alpha\theta_4}{\Gamma\theta_2\theta_3\theta_4 \vdash t(s) : \beta\theta_4} \tag{2}$$

The rule (2) corresponds to β-conversion of the ordinary λ-calculus [4].

2.2 Extended combinators

In this subsection, we introduce two combinators that enable us to change the order of λ-conversion, proposed by Steedman [6], as a kind of type change [3]. The ordinary λ-calculus requires

[1] θ_2, θ_3 are for $\Gamma\theta_2 \vdash t : \alpha \rightarrow \beta$ and for $\Gamma\theta_3 \vdash s : \alpha$, respectively. θ_4 unifies α which appears in both type declarations.

a strict order of conversion. However, in a concurrent model, this kind of strict order is a hindrance and contingent conversions are required.

C-combinator changes the order of λ-variables as follows:

$$\mathbf{C}(\lambda xy.f(x,y)) = \lambda yx.f(x,y).$$

Another requirement for exchanges of the order of λ-conversion is the following case. Suppose that we are required to compose all the following typed objects:

$$\begin{cases} f : \beta \rightarrow \gamma \\ g : \alpha \rightarrow \beta \\ a : \alpha \end{cases}$$

In such a case, we need to concatenate g and a first, and then $g(a)$ becomes applicable to f. However, with the help of the following **B**-combinator:

$$\mathbf{B}(\lambda x.f(x))(\lambda y.g(y)) = \lambda x.f(g(x)).$$

The λ-variable in g can be shifted beyond the scope of f so that we can concatenate f and g first, and, thus, have a become applicable as in Fig. 1.

$$\cfrac{\cfrac{\lambda y.g(y) \qquad a}{g(a)} \qquad \lambda x.f(x)}{f(g(a))}$$

$$\Downarrow$$

$$\cfrac{\mathbf{B}\,\cfrac{\lambda x.f(x) \qquad \lambda y.g(y)}{\lambda y.f(g(y))} \qquad a}{f(g(a))}$$

Figure 1: **B**-combinator

2.3 Cost of unification

The repeated use of **C**- and **B**-combinators is still problematic if we consider implementing it as an actual system because the termination of processing is not guaranteed. We have modeled the process of a partial decomposition as an abstraction of an argument of the first-order term. If this abstraction occurs randomly, the process easily falls into a loop. In order to avoid this, we assume the unification cost. If a compound term (a subtree) were to be decomposed once, the element with the longer distance should be abstracted first. We can regard the whole sentence structure as more grammatical if the sum of these unification costs is smaller. We introduce the heuristic costs [7], considering the parallelism between syntactic cases and semantic roles, as follows:

$$\begin{array}{ll} \|\theta_{(nom,agt)}\| = 1 & \|\theta_{(obj,nom)}\| = k \\ \|\theta_{(dat,cgt)}\| = 1 & \|\theta_{(dat,agt)}\| = k \\ \|\theta_{(acc,obj)}\| = 1 & \|\theta_{(t,obj)}\| = k \end{array}$$

$$\begin{array}{l} \|\theta_{(nom,dat)}\| = \infty \\ \|\theta_{(agt,cgt)}\| = \infty \\ \quad\vdots \end{array}$$

where $\theta_{(x,y)}$ represents a unifier of two DAG's: one's syntactic case is x and the other's semantic role is y. k is some constant larger than 1 ($k > 1$).

3 Chemical Abstract Machine

Chemical Abstract Machine (ChAM, for short) [1] is a paradigm of concurrent λ-calculus. In this paper, we will mention our principles on natural language processing with regard to the ChAM model.

We assume the process of natural language recognition as follows. Whenever a linguistic object is recognized, it is thrown into the *solution* of ChAM, and acts as a *molecule*. Verbs and some other auxiliary verbs introduces *membranes*. These membranes becomes their scopes for case (or role) domination; namely, each verb searches for molecules (noun phrases) that are necessary to satisfy each verb's case (role) frame, within its membrane. In some occasions, if multiple verbs exist in one sentence, they may conflict as to which verb dominates which noun phrase. In such a case, two membranes can interact and can exchange some molecules.

We use $s_1, s_2, s_3, \cdots$ for membranes. When a membrane s_i contains a molecule α, we denote as $s_i \models \alpha$. The supporting relation ($\models$) can be interpreted as an inclusion relation ($\supset$) in this case. Two membranes can interact when they contact with the notation '$\|$', as $s_1 \| s_2$. If there is a floating molecule (that which is not yet concatenated with other molecules) on one side, it can move through the porous membranes. Valences for concatenation of each molecule are represented by typed *lambda*-variables. If one membrane contains only one composite structure, and it still has surplus valences, we can regard that whole the membrane has those surplus valences as follows.

$$\begin{array}{c} s_2 \models \lambda xyz.make(x,y,z) \\ \downarrow \\ \lambda xyz.s_2 \models make(x,y,z) \end{array}$$

Now, we will apply our notions above to the actual problem of sentence generation.

(*yom-* = read)	Who reads?
Ken-wa Naomi-ni hon-wo yom-u.	*Ken*
Ken-wa Naomi-ni hon-wo yom-ase-ru.	*Naomi*
Ken-wa Naomi-ni hon-wo yom-are-ru.	*Naomi*
Ken-wa Naomi-ni hon-wo yom-ase-(r)-are-ru.	*Ken*

-wa	: nominative case marker
-ni	: dative case marker
-wo	: accusative case marker
hon	: noun for 'book'
yom-	: root of verb 'read'
-ase-	: auxiliary verb for causative
-are-	: auxiliary verb for passive
-(r)u	: present tense marker

Table 1: Agents alternation by agglutination of auxiliary verbs

4 Example: Japanese causative sentence

In the Japanese language, the causative and the change of voice are realized by agglutinations of those auxiliary verbs at the tail of current verbs. These auxiliary verbs as well as ordinary verbs can dominate some cases so that these agglutinations may change the whole syntax [9]. Namely the scope of the operation of these auxiliary verbs is not the operated verb but the whole sentence. In order to illustrate these role changes, we show the alternation of the agent of the main verb in Table 1 with a short tip to Japanese lexicon.

As an example, we will take the sentence:

> *Ken-wa Naomi-ni hon-wo yom-aseru.*
> (Ken makes Naomi read the book.)

First, we give DAG's for each lexical items in Fig 2. The last DAG in Fig. 2 represents that the verb '*yomu* (read)' requires two roles 'the reader' and 'the object to be read', and one optional role 'the counter-agent' who hears what the reader reads. In that figure, '$W \models$' means that each word is recognized in the general *world* however a verb '*yomu*' introduced a special membrane s_1 as a subworld of W. Each DAG means a polymorphic type of the lexical item.

Assume that there is a parser that constructs partial tree structures, as recognizing each word from the head sequentially. Then, when the first four words are recognized, they can form a complete sentence of (3).

$$s_1 \models \{read(K|_{\theta_1}, N|_{\theta_2}, B|_{\theta_3}) : \begin{bmatrix} cat & S \end{bmatrix}\} \qquad (3)$$

Because all the three nouns are adequately concatenated by 'read', a sentential representation is made in the subworld of s_1. In (3), θ_i's are the records of unification, that contain the costs and the original types; they become necessary when they are backtracked, and in that meaning, those bindings are transitive.

Now, let us recapitulate what has occurred in the membrane s_1. There were four lexical items in the set, and they are duly organized to a sentence and s_1 becomes a singleton.

$$\begin{array}{rcl} s_1 & = & \{K : N,\ N : N,\ B : N, \\ & & \lambda xyz.read(x,y,z) : N \rightarrow N \rightarrow N \rightarrow S\} \\ & & \downarrow \\ s_1 & = & \{read(K,N,B)\} \end{array}$$

Then, the problematic final word '*-aseru* (causative)' arrives; its DAG representation is as in Fig. 3. The DAG in Fig. 3 requires a sentential form (category S) as an argument, and in addition, it subcategorizes an item of category N as an agent of the subsentence.

Now, the process becomes as in Fig. 4. All through the process in Fig. 4, **C**- and **B**-combinators are used repeatedly as well as ordinary type inference (1) and (2). The second membrane s_2 requires an *agent* role (the variable x' of *make*). There is a record in θ_1 that it bit *agent*, so that the comparison should be made between θ_1 and $\theta_4 (= \theta_{(K,x')})$. However, because both of θ_1 and θ_4 unifies nominative case and agent role, the costs are equivalent. In such a case, linguistic heuristics will solve the problem. In this case, the agent of *make* should be the nominative of the whole sentence, and the co-agent of *make* is the dative of the whole sentence, so that K and N are bit by newly arrived *make*. B remains bound to *read*, because there is no λ-variable of that type in *make*. The process is depicted in fig. 5.

$$W \models K(= \mathit{Ken\text{-}wa}) : \begin{bmatrix} \mathit{cat} & N \\ \mathit{case} & \mathit{nom} \end{bmatrix}$$

$$W \models N(= \mathit{Naomi\text{-}ni}) : \begin{bmatrix} \mathit{cat} & N \\ \mathit{case} & \mathit{dat} \end{bmatrix}$$

$$W \models B(= \mathit{hon\text{-}wo}) : \begin{bmatrix} \mathit{cat} & N \\ \mathit{case} & \mathit{acc} \end{bmatrix}$$

$$s_1 \models \lambda xyz.\mathit{read}(x, y, z)(= \mathit{yom\text{-}}) : \begin{bmatrix} \mathit{val} & \begin{bmatrix} \mathit{cat} & S \\ \mathit{form} & \begin{bmatrix} \mathit{form} & \mathit{fintie} \end{bmatrix} \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{agent} \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{object} \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{co\text{-}agent} \\ \mathit{optionality} & + \end{bmatrix} \end{bmatrix}$$

Figure 2: Initial DAG

$$\lambda xyz.\mathit{make}(x, y, z(y)) : \begin{bmatrix} \mathit{val} & \begin{bmatrix} \mathit{cat} & S \\ \mathit{form} & \begin{bmatrix} \mathit{form} & \mathit{fintie} \end{bmatrix} \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{agent} \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{co\text{-}Agent} \\ \mathit{optionality} & + \end{bmatrix} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat} & S \\ \mathit{subcat} & \begin{bmatrix} \mathit{cat} & N \\ \mathit{role} & \mathit{agent} \end{bmatrix} \\ \mathit{role} & t \end{bmatrix} \end{bmatrix}$$

Figure 3: DAG for '*make*'

$$\begin{array}{rcl} s_1 \models \mathit{read}(K|_{\theta_1}, N|_{\theta_2}, B|_{\theta_3}) & \| & \lambda x'y'z'.s_2 \models \mathit{make}(x', y', z') \\ & \downarrow & \\ \lambda x.s_1 \models \mathit{read}(x, N|_{\theta_2}, B|_{\theta_3}) & \| & \lambda y'z'.s_2 \models \mathit{make}(K|_{\theta_4}, y', z') \\ & \downarrow & \\ \lambda xy.s_1 \models \mathit{read}(x, y, B) & \| & \lambda z'.s_2 \models \mathit{make}(K|_{\theta_4}, N|_{\theta_5}, z') \\ & \downarrow & \\ s_1 & \| & s_2 \models \mathit{make}(K|_{\theta_4}, N|_{\theta_5}, \lambda y.\mathit{read}(N, y, B)) \end{array}$$

Figure 4: Process

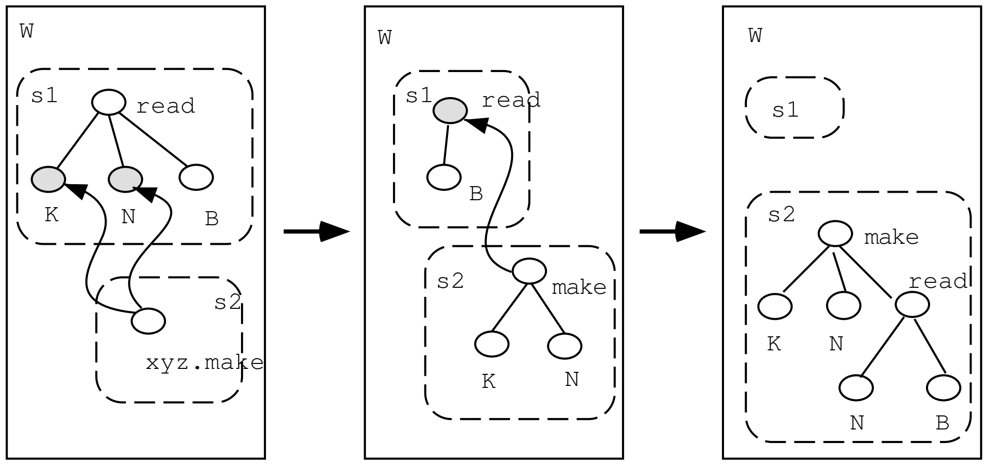

Figure 5: Membranes interaction

5 Conclusion

Introducing free-ordered typed λ-calculus, together with the notion of unification costs in types, we have shown the structuring of natural language syntax, by distributively represented types in random orders. We adopted a model of *Chemical Abstract Machine* for the partial/ concurrent computation model.

Although we introduced the concept of costs and termination was assured, the efficiency of constructing a parsing tree would be far slower than sequential processing. However our objective is not to propose a faster algorithm, but is to show the possibility of distributed processing of natural languages. We could show that natural language syntax is self-organizable, in that each linguistic objects do not need to be poured into 'molds', viz., externally given grammar.

References

[1] G. Berry and G. Boudol. The chemical abstract machine. In *17th Annual ACM Symposium on Principles of Programming Languages*, pages 81–93, 1990.

[2] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland, Amsterdam, Netherlands, 1968.

[3] D. Dowty. Type Raising, Functional Composition, and Non-constituent Conjunction - in *Categorial Grammars and Natural Language Structures*, pages 153–197. D. Reidel, 1988.

[4] J. R. Hindley and Seldin J. P. *Introduction to Combination and λ-Calculus.* Cambridge University Press, 1986.

[5] S. M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* CSLI, Stanford University, 1986.

[6] M. Steedman. Combinators and grammars - in *Categorial Grammars and Natural Language Structures*, pages 417–442. D. Reidel, 1988.

[7] S. Tojo. Categorial analysis of sentence generation. In *The Logic Programming Conference (LPC '91)*, pages 229–238. Institute of New Generation Computer (ICOT), 1991.

[8] H. Uszkoreit. Categorial unification grammars. In *Proc. of COLING '86*, pages 187–194, 1986.

[9] T. Gunji. *Japanese Phrase Structure Grammar.* D. Reidel, 1987.